CSCI403: Database Management

ENHANCING DATA ANALYSIS AND SCORING SYSTEMS FOR THE SOLAR CAR CHALLENGE FOUNDATION: A COMPREHENSIVE ANALYSIS OF REAL-WORLD DATA

submitted to

Christopher Painter-Wakefield

Colorado School of Mines

April 24, 2024

By

Brandon Ching

1 Background and Overview

The aim of this project and subsequent report is to analyze a real-world dataset using the principles and techniques learned in the CSCI403 Database Management course. Specifically, the dataset under scrutiny comprises event logs from the 2022 Solar Car Challenge. Utilizing SQL queries, we intend to extract and scrutinize data from this set, identifying trends, patterns, and insights that could enhance future iterations of the event. The report will encompass an overview of the dataset, details of the analysis performed, and a summary of findings.

I chose this dataset due to my involvement as a volunteer and judge in the Solar Car Challenge. Having contributed to the development of the timing and scoring system used during the event, I continue to collaborate with the organization to refine this system. The analyses conducted herein will inform and influence future events, including the upcoming 2024 edition.

2 About the Dataset

The dataset under examination is a proprietary collection provided by The Solar Car Challenge Foundation (SCCF). As a non-profit organization, SCCF hosts the Solar Car Challenge—an annual event where high school students from around the world design, build, and race solar vehicles. The dataset comprises event logs from the 2022 challenge, containing scoring and timing data for each participating team.

The data was extracted by retrieving CSV files exported from a PostgreSQL database post-event. These files generated two primary tables: `score_rawclickeydata` and `score_diary`. The former records "mark lap" events logged by the timing and scoring system, while the latter documents events noted by judges during the challenge. Both tables are linked by a common field, `team_id`, which serves as a unique identifier for each participating team.

While the archived dataset contained additional tables detailing participating teams, judges, and event metadata, the analysis primarily focused on scoring and timing data. These supplementary tables provided contextual information but were not integral to the analysis.

2.1 score_rawclickeydata Table

As shown in Table 1, the `score_rawclickeydata` table contains the following fields:

- **id:** Unique identifier for each record.
- **day:** Integer value representing the day of the event (1-4).

- **team_id:** ID of the team that the record pertains to.
- **time_stamp:** UTC timestamp of the event.
- **judge:** Name of the judge who recorded the event.

id	day	team_id	time_stamp	judge
3910	1	1	2022-07-17 13:03:32.454000	lsjaye-Brandon
3912	1	1	2022-07-17 13:03:32.685000	8hi33y-Liam
3913	1	1	2022-07-17 13:03:33.269000	8wk342-Michael1
3914	1	1	2022-07-17 13:05:43.315000	8wk342-Michael1
3915	1	1	2022-07-17 13:05:43.294000	c2dvxj-Ricardo
3919	1	2	2022-07-17 13:05:57.428000	c2dvxj-Ricardo
3920	1	2	2022-07-17 13:05:57.446000	lsjaye-Brandon
3921	1	2	2022-07-17 13:05:57.388000	8wk342-Michael1
3922	1	5	2022-07-17 13:08:07.675000	lsjaye-Brandon
3923	1	5	2022-07-17 13:08:07.932000	8hi33y-Liam

Table 1: score_rawclickeydata table sample data.

2.2 score_diary Table

As shown in Table 2, the `score_diary` table contains the following fields:

- **id:** Unique identifier for each record.
- **day:** Integer value representing the day of the event (1-4).
- **team_id:** ID of the team that the record pertains to.
- **time_stamp:** UTC timestamp of the event.
- **judge:** Name of the judge who recorded the event.
- **event:** Description of the event recorded (e.g., "mark lap", "enter track", etc).
- **num_passengers:** Number of passengers in the vehicle at the time of the event.

Id	Day	time_stamp	judge	event	num_passengers
20	1	2022-07-17 13:00:50.434000	igwgh3-Adayr	Enter Track	<null>
21	2	2022-07-17 13:01:26.072000	igwgh3-Adayr	Enter Track	<null>
22	5	2022-07-17 13:03:26.401000	igwgh3-Aldayr	Enter Track	<null>
23	7	2022-07-17 13:04:25.393000	igwgh3-Aldayr	Enter Track	<null>
24	8	2022-07-17 13:05:24.882000	igwgh3-Aldayr	Enter Track	<null>
25	9	2022-07-17 13:06:24.519000	igwgh3-Aldayr	Enter Track	<null>
26	10	2022-07-17 13:07:26.263000	igwgh3-Aldayr	Enter Track	<null>
27	11	2022-07-17 13:08:24.447000	6c8fwd-Lucas	Enter Track	<null>
28	11	2022-07-17 13:08:45.302000	6c8fwd-Lucas	Exit Track	<null>
29	12	2022-07-17 13:09:25.215000	6c8fwd-Lucas	Enter Track	<null>

Table 2: score_diary table sample data.

2.3 Data Collection Method

The dataset was compiled using a custom-built timing and scoring system tailored for the Solar Car Challenge. This system features a bespoke web interface enabling judges to record team lap completions and predefined events. Typically, a minimum of three judges mark lap completions and events, resulting in at least three separate records for each valid lap.

2.4 License and Privacy

This dataset remains the property of SCCF and should not be shared or distributed without their explicit permission. To protect participant privacy, any personally identifiable information has been removed from the dataset.

3 Data Analysis

During the data analysis phase, 4 primary queries were developed to extract and analyze data from the dataset. These queries aimed to provide insights into the performance of participating teams, identify trends, and refine the scoring system. The queries were designed to calculate the total lap credit earned by each team on a given day, along with the average and best lap times for each team. Additionally, the queries determined the total laps completed by each team, accounting for penalties and bonus laps. The results were then sorted by division and day's lap credit, with the leading team in each division assigned a rank of 1. The queries also aggregated supplementary data such as team names and websites to provide context to the results.

These queries were:

1. **Audit Query:** Provide a detailed data dump of the scoring and timing data. The output of this query allows for a comprehensive review of the data, ensuring the accuracy and integrity of the scoring system. This query is the backbone of the scoring system and is referenced by all following queries.

2. **Live Scoring Query:** Process the raw data in real time, allowing for the results of the query to be displayed on a live web interface. This query provides real-time unofficial results to participants and spectators, enhancing the event experience.

3. **Specific Team Lap Stats:** For a specific team, calculate the lap times for a specified day of the event. This query provides detailed insights into the performance of individual teams, allowing for targeted analysis and feedback. This also has the added ability to plot teams lap times over the course of the event.

4. **Judge accuracy Query:** Analyze the data to determine the accuracy of individual judges. This query calculates the average deviation of each judge's lap times from the mean, providing insights into the consistency and reliability of each judge.

NOTE: Most of the queries developed for this analysis ended up being extremely long and complicated. I apologies for them being hard to read and follow.

3.1 Audit Query

At the core of the scoring system is the audit query, which provides a detailed data dump of the scoring and timing data. This query is saved as a view as `live_rawscore` in the database allowing the data to be easily accessed and reviewed. The audit query outputs the following fields:

- **id:** A unique identifier for each record.[1]
- **day:** Integer value representing the day of the event (1-4).
- **team_id:** ID of the team that the record pertains to.
- **time_stamp:** UTC timestamp of the event.
- **judge:** Name of the judge who recorded the event.
- **event:** Description of the event recorded (e.g., "mark lap", "enter track", etc)
- **num_passengers:** Number of passengers in the vehicle at the time of the event.
- **passengers_change_timestamp:** Timestamp of the last change of passengers.
- **any_event_prev_time_diff:** Time difference between the current event and the previous event.
- **any_event_next_time_diff:** Time difference between the current event and the next event.
- **same_event_prev_time_stamp:** Timestamp of the previous event of the same type.
- **same_event_prev_time_diff:** Time difference between the current event and the previous event of the same type.
- **same_event_prev_2_time_diff:** Time difference between the current event and the event before the previous event of the same type.
- **same_event_next_time_stamp:** Timestamp of the next event of the same type.
- **same_event_next_time_diff:** Time difference between the current event and the next event of the same type.

[1] This field is not a key due to the fact it displays ids from both the `score_rawclickeydata` and `score_diary` tables. Each of these tables had their own unique id auto incrementing field. This results in some overlapping ids. This issue has since been resolved in the current version of the database that will be used for 2024.

- **same_event_next_2_time_diff:** Time difference between the current event and the event after the next event of the same type.
- **same_event_same_judge_next_time_diff:** Time difference between the current event and the next event of the same type recorded by the same judge.
- **last_time_entered_track:** Timestamp of the last time the team entered the track.
- **last_time_exited_track:** Timestamp of the last time the team exited the track.
- **current_location:** Description of the current location of the team (e.g., "on track" or "off track").
- **confirmed_lap:** Boolean value indicating whether the lap was confirmed.
- **lap_click_count:** Number of times the lap was clicked.
- **lap_click_any:** Boolean value indicating whether the lap was clicked at all.
- **self_confirmed_lap:** Boolean value indicating whether the lap was self-confirmed.
- **num_passengers_impute:** Number of passengers imputed by the system.
- **lap_time:** Time taken to complete the lap.
- **low_confidence_lap:** Description of the confidence level of the lap.

Due to the number of fields and the complexity of the query, a sample of the query results is not provided in this report. However, the full query is included in Listing 1.

```sql
SELECT raw_and_diary_2.id,
    raw_and_diary_2.day,
    raw_and_diary_2.team_id,
    raw_and_diary_2.time_stamp,
    raw_and_diary_2.judge,
    raw_and_diary_2.event,
    raw_and_diary_2.num_passengers,
    raw_and_diary_2.passengers_change_timestamp,
    raw_and_diary_2.any_event_prev_time_diff,
    raw_and_diary_2.any_event_next_time_diff,
    raw_and_diary_2.same_event_prev_time_stamp,
    raw_and_diary_2.same_event_prev_time_diff,
    raw_and_diary_2.same_event_prev_2_time_diff,
    raw_and_diary_2.same_event_next_time_stamp,
    raw_and_diary_2.same_event_next_time_diff,
    raw_and_diary_2.same_event_next_2_time_diff,
    raw_and_diary_2.same_event_same_judge_next_time_diff,
    raw_and_diary_2.last_time_entered_track,
    raw_and_diary_2.last_time_exited_track,
    raw_and_diary_2.current_location,
    raw_and_diary_2.confirmed_lap,
    raw_and_diary_2.lap_click_count,
    raw_and_diary_2.lap_click_any,
    raw_and_diary_2.self_confirmed_lap,
    raw_and_diary_2.num_passengers_impute,
    CASE
      WHEN raw_and_diary_2.num_passengers_impute IS NOT NULL
        THEN
          raw_and_diary_2.num_passengers_impute *
          raw_and_diary_2.confirmed_lap
      ELSE raw_and_diary_2.confirmed_lap
      END AS lap_credit,
    CASE
      WHEN raw_and_diary_2.lap_click_any >=
           1 AND
           raw_and_diary_2.same_event_prev_time_stamp >=
           COALESCE(
               raw_and_diary_2.last_time_entered_track,
               ’2021-01-01 00:00:00’::timestamp without time zone)
        THEN
          raw_and_diary_2.time_stamp -
          raw_and_diary_2.same_event_prev_time_stamp
      WHEN raw_and_diary_2.lap_click_any >=
           1 AND
           COALESCE(
               raw_and_diary_2.same_event_prev_time_stamp,
               ’2021-01-01 00:00:00’::timestamp without time zone) <
           raw_and_diary_2.last_time_entered_track
        THEN
          raw_and_diary_2.time_stamp -
          raw_and_diary_2.last_time_entered_track
      ELSE ’00:00:00’::interval
      END AS lap_time,
    CASE
      WHEN
          raw_and_diary_2.lap_click_count =
          1
        THEN 'single click lap'::text
      WHEN raw_and_diary_2.lap_click_count =
           2 AND
           raw_and_diary_2.self_confirmed_lap =
           1
        THEN 'self confirmed lap'::text
      WHEN raw_and_diary_2.lap_click_count >=
           2 AND
           (raw_and_diary_2.current_location <> ALL
            (ARRAY ['on track'::text, 'leaving track'::text]))
        THEN 'check car location'::text
      ELSE ''::text
      END AS low_confidence_lap
FROM (SELECT raw_and_diary_1.id,
          raw_and_diary_1.day,
          raw_and_diary_1.team_id,
          raw_and_diary_1.time_stamp,
          raw_and_diary_1.judge,
          raw_and_diary_1.event,
          raw_and_diary_1.num_passengers,
          raw_and_diary_1.passengers_change_timestamp,
          raw_and_diary_1.any_event_prev_time_diff,
          raw_and_diary_1.any_event_next_time_diff,
          raw_and_diary_1.same_event_prev_time_stamp,
          raw_and_diary_1.same_event_prev_time_diff,
          raw_and_diary_1.same_event_prev_2_time_diff,
          raw_and_diary_1.same_event_next_time_stamp,
          raw_and_diary_1.same_event_next_time_diff,
          raw_and_diary_1.same_event_next_2_time_diff,
          raw_and_diary_1.same_event_same_judge_next_time_diff,
          raw_and_diary_1.last_time_entered_track,
          raw_and_diary_1.last_time_exited_track,
          CASE
            WHEN
                raw_and_diary_1.last_time_entered_track >
                raw_and_diary_1.last_time_exited_track
```

```
              THEN 'on track'::text
            WHEN
                raw_and_diary_1.last_time_entered_track IS NOT NULL AND
                raw_and_diary_1.last_time_exited_track IS NULL
              THEN 'on track'::text
            WHEN raw_and_diary_1.last_time_entered_track <
                 raw_and_diary_1.last_time_exited_track AND
                 (raw_and_diary_1.time_stamp -
                  raw_and_diary_1.last_time_exited_track) <=
                 '00:00:05'::interval
              THEN 'leaving track'::text
            WHEN
                raw_and_diary_1.last_time_entered_track <
                raw_and_diary_1.last_time_exited_track
              THEN 'in garage'::text
            ELSE 'unknown location'::text
            END                         AS current_location,
          CASE
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 raw_and_diary_1.same_event_next_time_diff <=
                 '00:00:30'::interval AND
                 COALESCE(
                     raw_and_diary_1.same_event_prev_time_diff,
                     '00:10:00'::interval) >=
                 '00:00:31'::interval
              THEN 1
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 raw_and_diary_1.same_event_next_time_diff <=
                 '00:00:30'::interval AND
                 raw_and_diary_1.same_event_prev_time_diff IS NULL
              THEN 1
            ELSE 0
            END                         AS confirmed_lap,
          CASE
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 COALESCE(
                     raw_and_diary_1.same_event_prev_time_diff,
                     '00:10:00'::interval) >=
                 '00:00:31'::interval AND
                 raw_and_diary_1.same_event_next_time_diff <=
                 '00:00:30'::interval AND
                 raw_and_diary_1.same_event_next_2_time_diff <=
                 '00:00:30'::interval
              THEN 3
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 COALESCE(
                     raw_and_diary_1.same_event_prev_time_diff,
                     '00:10:00'::interval) >=
                 '00:00:31'::interval AND
                 raw_and_diary_1.same_event_next_time_diff <=
                 '00:00:30'::interval
              THEN 2
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 COALESCE(
                     raw_and_diary_1.same_event_prev_time_diff,
                     '00:10:00'::interval) >=
                 '00:00:31'::interval
              THEN 1
            ELSE 0
            END                         AS lap_click_count,
          CASE
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 COALESCE(
                     raw_and_diary_1.same_event_prev_time_diff,
                     '00:10:00'::interval) >=
                 '00:00:31'::interval
              THEN 1
            ELSE 0
            END                         AS lap_click_any,
          CASE
            WHEN lower(raw_and_diary_1.event::text) =
                 'mark lap'::text AND
                 raw_and_diary_1.same_event_same_judge_next_time_diff <=
                 '00:00:30'::interval
```

```
               THEN 1
             ELSE 0
             END                        AS self_confirmed_lap,
           num_pass_diary.num_passengers AS num_passengers_impute
    FROM (SELECT raw_and_diary.id,
                 raw_and_diary.day,
                 raw_and_diary.team_id,
                 raw_and_diary.time_stamp,
                 raw_and_diary.judge,
                 raw_and_diary.event,
                 raw_and_diary.num_passengers,
                 max(
                 CASE
                   WHEN
                       COALESCE(
                           raw_and_diary.num_passengers,
                           '-1'::integer) >
                       0
                     THEN raw_and_diary.time_stamp
                   ELSE NULL::timestamp without time zone
                   END)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   (EXTRACT(
                       day
                       FROM
                       raw_and_diary.time_stamp))
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id
                   ROWS UNBOUNDED PRECEDING) AS passengers_change_timestamp,
                 raw_and_diary.time_stamp -
                 lag(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id)       AS any_event_prev_time_diff,
                 lead(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id) -
                 raw_and_diary.time_stamp    AS any_event_next_time_diff,
                 lag(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id)       AS same_event_prev_time_stamp,
                 raw_and_diary.time_stamp -
                 lag(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id)       AS same_event_prev_time_diff,
                 raw_and_diary.time_stamp -
                 lag(
                 raw_and_diary.time_stamp,
                 2)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id)       AS same_event_prev_2_time_diff,
                 lead(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id)       AS same_event_next_time_stamp,
                 lead(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id) -
                 raw_and_diary.time_stamp    AS same_event_next_time_diff,
                 lead(
                 raw_and_diary.time_stamp,
                 2)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id) -
                 raw_and_diary.time_stamp    AS same_event_next_2_time_diff,
                 lead(
                 raw_and_diary.time_stamp,
                 1)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   raw_and_diary.event,
                   raw_and_diary.judge
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id) -
                 raw_and_diary.time_stamp    AS same_event_same_judge_next_time_diff,
                 max(
                 CASE
                   WHEN
                       lower(raw_and_diary.event::text) =
                       'enter track'::text
                     THEN raw_and_diary.time_stamp
                   ELSE NULL::timestamp without time zone
                   END)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   (EXTRACT(
                       day
                       FROM
                       raw_and_diary.time_stamp))
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id ROWS UNBOUNDED PRECEDING)
                                             AS last_time_entered_track,
                 max(
                 CASE
                   WHEN
                       lower(raw_and_diary.event::text) =
                       'exit track'::text
                     THEN raw_and_diary.time_stamp
                   ELSE NULL::timestamp without time zone
                   END)
                 OVER (PARTITION BY raw_and_diary.team_id,
                   (EXTRACT(
                       day
                       FROM
                       raw_and_diary.time_stamp))
                   ORDER BY raw_and_diary.time_stamp,
                     raw_and_diary.id ROWS UNBOUNDED PRECEDING)
                                             AS last_time_exited_track
          FROM (SELECT score_rawclickeydata.id,
                       score_rawclickeydata.day,
                       score_rawclickeydata.team_id,
                       score_rawclickeydata.time_stamp,
                       score_rawclickeydata.judge,
                       'mark lap'::character varying AS event,
                       '-1'::integer                  AS num_passengers
                FROM score_rawclickeydata
                UNION ALL
                SELECT score_diary.id,
                       score_diary.day,
                       score_diary.team_id,
                       score_diary.time_stamp,
                       score_diary.judge,
                       score_diary.event,
                       score_diary.num_passengers
                FROM score_diary) raw_and_diary
          ORDER BY raw_and_diary.team_id,
                   raw_and_diary.time_stamp,
                   raw_and_diary.id) raw_and_diary_1
           LEFT JOIN (SELECT score_diary.time_stamp,
                             score_diary.team_id,
                             score_diary.num_passengers
                      FROM score_diary) num_pass_diary
                     ON num_pass_diary.time_stamp =
                        raw_and_diary_1.passengers_change_timestamp AND
                        raw_and_diary_1.team_id =
                        num_pass_diary.team_id
    ORDER BY raw_and_diary_1.team_id,
             raw_and_diary_1.time_stamp,
             raw_and_diary_1.id) raw_and_diary_2
 ORDER BY raw_and_diary_2.team_id,
       raw_and_diary_2.time_stamp,
       raw_and_diary_2.id;
```

Listing 1: Full Audit query

3.2 Live Scoring Query

The primary objective in analyzing this dataset was to refine the precision and efficiency of the scoring system. Previously, achieving this involved labor-intensive manual review to categorize similar events and ensure each category had multiple validating records. This process was time-consuming and prone to error. The new query, as detailed in Listing 2, computes the total lap credit earned by each team on a given day, along with the average and best lap times for each team. Additionally, it determines the total laps completed by each team, accounting for penalties and bonus laps. The resulting data is then sorted by division and day's lap credit, with the leading team in each division assigned a rank of 1. The query also aggregates supplementary data such as team names and websites to provide context to the results. A sample of the query results for day 3 of the 2021 Solar Car Challenge is presented in Table 3.

The resulting query outputs the following fields:

- **rank:** Rank of the team within its division based on the day's lap credit.
- **division:** Division of the team (numerical identifier which can be referenced against a different table not included in this analysis).
- **team_id:** ID of the team.
- **day_lap_credit:** Total lap credit earned by the team on the given day.
- **average_lap_time:** Average lap time for the team on the given day.
- **best_lap_time:** Best lap time for the team on the given day.
- **total_laps:** Total laps completed by the team, accounting for penalties and bonus laps.

```sql
-- This query calculates various statistics for teams participating
-- in a competition. Assigns a rank to each team within its division
-- based on the day's lap credit, ordered in descending order.
SELECT Rank() over (
    PARTITION BY score_team.division
    ORDER BY parsed_day_score.day_lap_credit DESC
    ) AS rank,
        score_team.division,
        parsed_day_score.team_id,
        parsed_day_score.day_lap_credit,
        parsed_day_score.average_lap_time,
        parsed_day_score.best_lap_time,
        total_laps.total_laps,
        score_team.team_name,
        score_team.team_website
FROM (
        -- Subquery to calculate statistics for each team for the day.
        SELECT live_rawscore.team_id,
                -- Calculates the total lap credit earned by each team.
                SUM(live_rawscore.lap_credit) AS day_lap_credit,
                -- Calculates the average lap time for each team.
                To_char(
                    Avg(
                        CASE
                        WHEN live_rawscore.lap_time >
                                '00:00:00'::interval
                            THEN live_rawscore.lap_time
                        ELSE NULL::interval
                        END
                    ),
                    'MI:SS'::text
                )                             AS average_lap_time,
                -- Calculates the best lap time for each team.
                To_char(
                    CASE
                    WHEN Min(
                        CASE
                            WHEN live_rawscore.lap_time >
                                '00:00:00'::interval
                            THEN live_rawscore.lap_time
                            ELSE NULL::interval
                            END
                            ) IS NULL THEN '00:00:00'::interval
                    ELSE Min(
                        CASE
                            WHEN live_rawscore.lap_time >
                                '00:00:00'::interval
                            THEN live_rawscore.lap_time
                            ELSE NULL::interval
                            END
                            )
```

```sql
                    END,
                    'MI:SS'::text
                )                             AS best_lap_time
        FROM live_rawscore
        WHERE live_rawscore.day = 3
        GROUP BY live_rawscore.team_id) parsed_day_score
        -- Joins the total laps calculation with the team details.
        join score_team
            ON score_team.id = parsed_day_score.team_id
        join (
    -- Subquery to calculate the total laps for each team, considering
    -- penalties and bonus laps.
    SELECT total_laps_1.team_id,
            total_laps_1.total_lap_credit +
            score_forever_laps.laps -
            score_penalty.laps AS total_laps
    FROM (
            -- Subquery to calculate the total laps for each team.
            SELECT live_rawscore.team_id,
                SUM(live_rawscore.lap_credit) AS total_lap_credit
            FROM live_rawscore
            WHERE live_rawscore.day <= 3
            GROUP BY live_rawscore.team_id) total_laps_1
            -- Joins the total laps with bonus laps.
            join score_penalty ON score_penalty.team_id =
                                total_laps_1.team_id
            join score_forever_laps
                ON score_forever_laps.team_id =
                    total_laps_1.team_id
    WHERE score_penalty.day <= 3) total_laps
            ON total_laps.team_id = score_team.id
-- Orders the results by division then by rank within each division.
ORDER BY score_team.division,
            (
                Rank() over (
                PARTITION BY score_team.division
                ORDER BY parsed_day_score.day_lap_credit DESC
                )
            );
```

Listing 2: Live Score Query for day 3

Table 3: Sample query results from Listing 2 for day 3 of the 2021 Solar Car Challenge.

rank	division	team_id	day_lap_credit	average_lap_credit	best_lap_time	total_laps
1	0	5	588	02:42	01:57	1090
2	0	11	208	01:53	00:40	678
3	0	9	110	03:20	02:28	333
4	0	1	93	04:03	02:24	236
5	0	7	66	05:25	03:17	145
1	1	12	104	03:31	02:24	356
2	1	16	78	04:15	02:12	235
3	1	18	76	04:24	00:41	256
4	1	19	73	04:54	01:15	238
5	1	21	70	04:46	03:13	202

3.3 Specific Team Lap Stats

The specific team lap stats query was designed to provide detailed insights into the performance of individual teams. This query calculates the lap times for a specific team on a specific day of the event. This allows for the data to be presented in a graphical format, such as shown in Figure 1. The query outputs the following fields:

- **time_stamp:** UTC timestamp of the event.
- **lap_time:** Time taken to complete the lap.

```sql
-- Selecting specific columns and formatting lap_time as MM:SS
SELECT time_stamp,
        TO_CHAR(lap_time, 'MI:SS') AS lap_time -- Format as MM:SS
FROM live_rawscore -- Table containing the raw lap scores
WHERE team_id = 11      -- Filtering rows where team_id is 11
    AND confirmed_lap = 1 -- Filtering rows where lap is confirmed
    AND day = 1           -- Filtering rows for day 1
ORDER BY time_stamp ASC;
```

Listing 3: Live Score Query for day 3

A sample of the output is provided in Table 4 and Figure 1. Only the timestamp and lap time are included in the results as this information is sufficient and will eventually be used in an API to display the data in a web based graphical format; this reduces the amount of data that needs to be transferred and processed.

time_stamp	lap_time
2022-07-17 13:20:44.164000	03:44
2022-07-17 13:23:15.328000	02:30
2022-07-17 13:25:33.828000	02:17
2022-07-17 13:27:50.141000	02:16
2022-07-17 13:30:04.127000	02:13
2022-07-17 13:32:17.549000	02:11

Table 4: Lap Times for Team 11

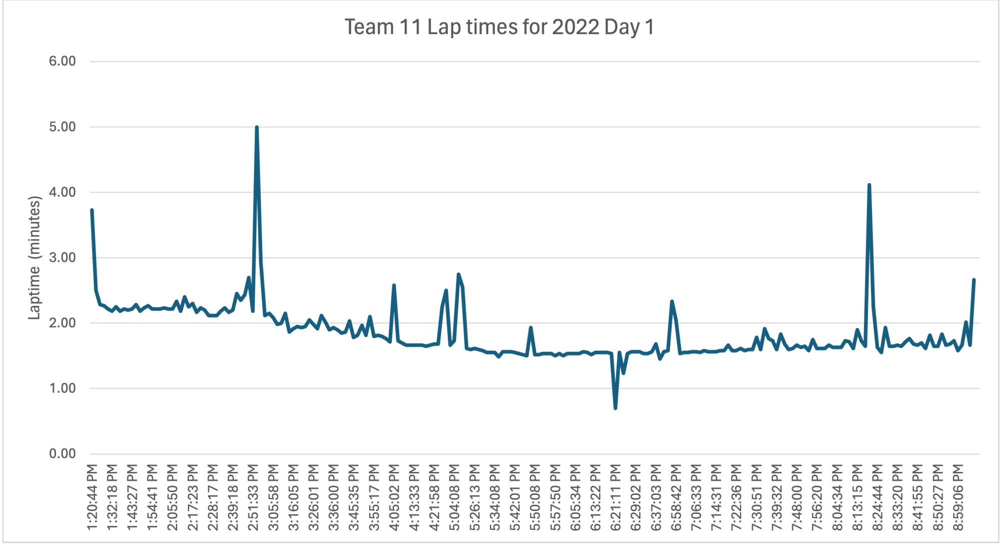

Figure 1: Lap Times for Team 11

3.4 Judge Accuracy Query

A key metric which is useful in selecting which judges are allowed to continue their role in scoring is the accuracy of their scoring. This query calculates the accuracy of each judge by determining the number of "bad events" recorded by each judge. Bad events are determined based on certain criteria:

- The time of the event is more than one standard deviation away from the average time of all records for the same team and time interval (lap).
- The time range of the events for the same team and time interval is greater than one second.

The query outputs the following fields:

- **judge_name:** Name of the judge.
- **bad_events:** Count of bad events recorded by the judge.
- **total_events:** Total count of events recorded by the judge.
- **bad_event_rate:** Rate of bad events recorded by the judge.

```sql
-- Calculate the count of "bad events" for each judge
-- Bad events are determined based on certain criteria
WITH judge_counts
        AS (SELECT subquery.judge_name,
                    COUNT(*) AS bad_events
            FROM (
                    -- Subquery to compute metrics and filter bad events
                    SELECT lr.*,
                            grp.avg_event_count,
                            TO_TIMESTAMP(grp.avg_time)      AS avg_time,
                            grp.std_dev_timestamp,
                            EXTRACT(
                                EPOCH
                                FROM
                                TO_TIMESTAMP(grp.max_time) -
                                TO_TIMESTAMP(grp.min_time)) AS time_range,
                            CASE
                            WHEN
                                grp.std_dev_timestamp !=
                                0
                                THEN
                                (EXTRACT(
                                    EPOCH
                                    FROM
                                    lr.time_stamp) -
                                grp.avg_time) /
                                grp.std_dev_timestamp
                            ELSE NULL
                            END                           AS num_std_devs,
                            REGEXP_REPLACE(
                                LOWER(SUBSTRING(
                                    lr.judge
                                    FROM
                                    POSITION(
                                        '_'
                                        IN
                                        lr.judge) +
                                    1)),
                                '\d',
                                '')                         AS judge_name
                    FROM score_rawclickeydata lr
                            JOIN (
                    -- Subquery to calculate aggregated metrics
                    SELECT team_id,
                            (EXTRACT(
                                EPOCH
                                FROM
                                time_stamp) /
                            30)::INT *
                            30               AS time_interval,
                            AVG(
                            COUNT(*))
                            OVER (PARTITION BY team_id,
                                (EXTRACT(
                                    EPOCH
                                    FROM
                                    time_stamp) /
                                30)::INT)     AS avg_event_count,
                            AVG(EXTRACT(
                                EPOCH
                                FROM
                                time_stamp)) AS avg_time,
                            STDDEV(EXTRACT(
                                EPOCH
                                FROM
                                time_stamp)) AS std_dev_timestamp,
                            MIN(EXTRACT(
                                EPOCH
                                FROM
                                time_stamp)) AS min_time,
                            MAX(EXTRACT(
                                EPOCH
                                FROM
                                time_stamp)) AS max_time
                    FROM score_rawclickeydata
                    GROUP BY team_id,
                                (EXTRACT(
                                    EPOCH
                                    FROM
                                    time_stamp) /
                                30)::INT) grp
                                ON lr.team_id =
                                    grp.team_id
                                AND
                                    (EXTRACT(
                                        EPOCH
                                        FROM
                                        lr.time_stamp) /
                                    30)::INT *
                                    30 =
                                    grp.time_interval
                    ORDER BY lr.team_id,
                            (EXTRACT(
                                EPOCH
                                FROM
                                lr.time_stamp) /
                            30)::INT *
                            30) AS subquery
            -- Filtering criteria for bad events
            WHERE ABS(subquery.num_std_devs) >
                    1
                AND subquery.time_range >
                    1
            GROUP BY subquery.judge_name),
-- Calculate the total count of events for each judge
        total_judge_counts
        AS (SELECT REGEXP_REPLACE(
                        LOWER(SUBSTRING(
                            raw_data.judge
                            FROM
                            POSITION(
                                '_'
                                IN
                                raw_data.judge) +
                            1)),
                        '\d',
                        '')  AS judge_name,
                    COUNT(*) AS total_events
            FROM score_rawclickeydata AS raw_data
            GROUP BY judge_name)
-- Join the counts of bad events and total events for each judge
-- Also calculate the bad event rate for each judge
SELECT judge_counts.judge_name,
        bad_events,
        total_events,
        CASE
            WHEN
                total_events !=
                0
            THEN
                CAST(bad_events AS FLOAT) /
                total_events
            ELSE 0
            END AS bad_event_rate
FROM judge_counts
        JOIN total_judge_counts
            ON judge_counts.judge_name =
                total_judge_counts.judge_name
-- Order the results by the count of bad events in descending order
```

```
ORDER BY bad_events DESC;
```

Listing 4: Query to calculate timing stats

Table 5: Sample query results from Listing 4.

judge_name	bad_events	total_events	bad_event_rate
Judge 1	429	3510	0.122
Judge 2	153	2666	0.057
Judge 3	110	3818	0.029
Judge 4	90	1238	0.073
Judge 5	81	490	0.165
Judge 6	77	1176	0.065
Judge 7	24	679	0.035
Judge 8	12	130	0.092
Judge 9	11	279	0.039
Judge 10	11	340	0.032

While not explicitly what the query was designed for, an interesting observation made from one of the subqueries while developing the query was the distribution of consistency in the time intervals between events. As shown in Figure 2, the overall distribution of time intervals between events is quite uniform, with a peak around 0 seconds. This indicates that most events are recorded in quick succession, which is expected given the nature of the competition. This also shows that over 90% of the events are recorded within 2 seconds of each other, which is a good sign of consistency in the data.

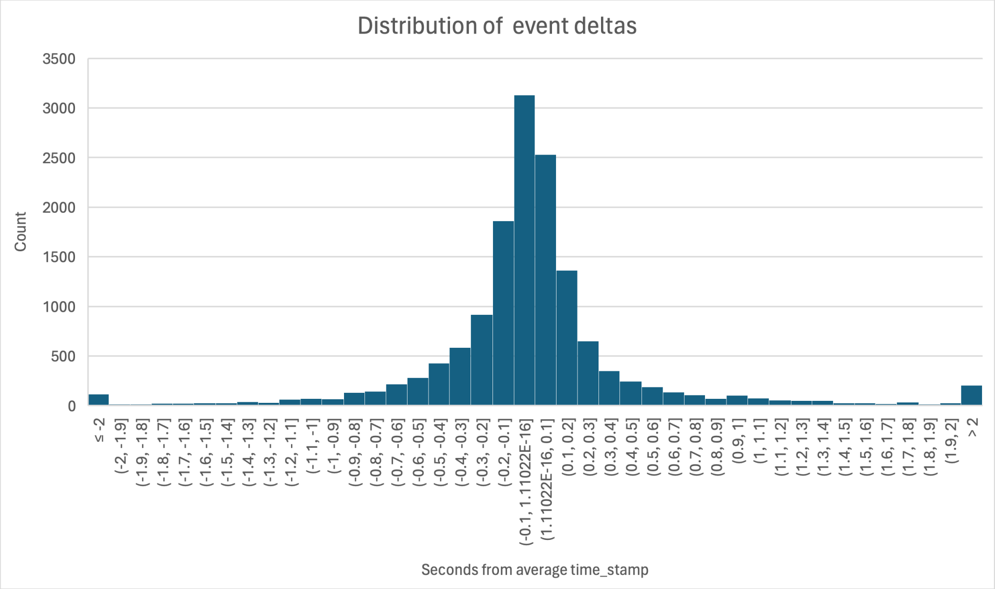

Figure 2: Distribution of time intervals between events.

4 Technical Challenges

The primary technical challenge encountered was a few missing tables from the archived dataset. Fortunately, these missing tables were not critical to the analysis and could be reconstructed from the available data. Reconstruction was conducted using publicly available information on the SCCF website, along with my knowledge of the event and some placeholder data.

Somewhat surprisingly, one of the biggest challenges was formatting the SQL queries to fit within the page margins for this report. Most of the queries were quite long and required significant effort to format correctly. Apologies for the weird formatting and somewhat hard to read queries.

5 Tools Used

The primary tool used in the analysis was PostgreSQL as the dataset was stored in a PostgreSQL database. Both VS Code and DataGrip were used in the writing of the queries, however all executions occurred in DataGrip as it provided the best UI for viewing the results. All graphs presented in this report were generated by exporting the results to CSV and imported into Excel for processing. The final report was written in LaTeX using VS Code (Because local development is superior! Sorry Overleaf).